Whispers in the Cloud
A Covert Channel using the Result of Creating a Virtual Machine

Cong Li, Qingni Shen, Kang Chen, Yahui Yang and Zhonghai Wu
School of Software and Microelectronics & MoE Key Lab of Network and Software Assurance, Peking University, Beijing, China

Keywords: Covert Channel, OpenStack, Virtual Machine.

Abstract: With the widespread use of cloud computing, people pay more attention to the security of cloud platforms. For the case of some clouds, users are permitted to use the services, but they cannot communicate with each other in the same cloud. In this paper, we present a new kind of user-level covert channel which we called CCRCVM (Covert Channel using the Result of Creating a Virtual Machine). This covert channel exists in OpenStack, which we have confirmed. This covert channel takes advantage of the result of creating a virtual machine to make the users communicate. First of all, we describe the threat scenario of this covert channel. Then, we describe the theory and communication process of the covert channel. Afterwards, we implement the covert channel in our own OpenStack environment. We also measure the bandwidth and communication accuracy of this covert channel in many times. Finally, we discuss how to mitigate and eliminate this channel.

1 INTRODUCTION

With the widespread use of cloud computing, people pay more attention to the security of cloud platforms. Thanks to the resource sharing, the resources can be used more efficiently. However, resource sharing is a double-edged sword. The information will be leaked because of the resource sharing. Isolation is designed to eliminate this kind of threat. It is still not strong enough (Reuben, 2007). Malicious users can still carry out the information leakage attack by covert channels.

Covert channel is generally referred to as a communication mechanism that is neither designed nor intended to transmit information (Lampson, 1973). It has been widely recognized as a serious threat to not only operating systems but also virtualized platforms (Jaeger and Sreenivasan, 2007).

In multi-user systems, covert channel is a well-known type of security attack. Originated in 1972 by Lampson (Lampson, 1973), the threats of covert channels are present in systems with shared resources, such as file system objects (Lampson, 1973), virtual memory (Vleck, 1990), processor caches (Percival, 2005), input devices etc. (Shah et al., 2006) (Meade, 1993).

The cloud system is a multi-user system, so the covert channel threat exists in cloud systems. According to the place where the channels communicate, we divide cloud covert channels into two categories: virtual machine level (VM-level) covert channel and user-level covert channel. The great majority of cloud covert channels are VM-level. There are some difficulties in these VM-level covert channels. The VMs use the shared resources to construct the covert channels. These cloud covert channels usually need VMs to be co-residency (Ristenpart et al., 2009) which means different VMs run on the same host. Many researchers have focused on this area (Varadarajan et al., 2015) (Bijon et al., 2015) (Han, Y et al., 2014). In addition, the VMs' behaviours are monitored closely by cloud providers (Pitropakis et al., 2015) (Alarifi and Wolthusen, 2012). Therefore, these covert channels are hard to use. Even so, the covert channel is still an important threat to cloud platforms.

In this paper, we present a novel covert channel called CCRCVM. This covert channel is a kind of user-level cloud covert channel. This covert channel happens between two users in the same cloud platform. It uses the *server-group* to bind the virtual machines to a physical host in each user account. The sender and receiver need to create VMs on the same host. The sender is able to influence the host state by making the host full-load. The receiver is able to get the bit by observing the result of creating a virtual

Li, C., Shen, Q., Chen, K., Yang, Y. and Wu, Z.
Whispers in the Cloud - A Covert Channel using the Result of Creating a Virtual Machine.
DOI: 10.5220/0005739403800387
In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pages 380-387
ISBN: 978-989-758-167-0

machine. We also design the synchronization process to transmit the bit correctly. We implement the covert channel in our own OpenStack environment. We do many experiments in this environment. The results show that we can transmit the information with the 100% accuracy when the bandwidth is 0.0167bps (bit per second). We also discuss how to mitigate and eliminate this channel.

Our contributions are as follows. (1) We present a novel kind of user-level covert channel called CCRCVM in cloud platforms. To the best of our knowledge, this paper is the first cloud covert channel to be used in user-level. (2) We analyse the theory of CCRCVM and implement a prototype in OpenStack. (3) We discuss the relationship between the bandwidth and the accuracy, and give the way to mitigate and eliminate this channel.

The remainder of this paper is organized as follows. Section 2 describes the related work of the covert channel. Section 3 describes the threat scenario and communication theory. Section 4 describes the implementation of CCRCVM. Section 5 evaluates the bandwidth and accuracy. Section 6 discusses the working conditions and countermeasures. Section 7 concludes this paper.

2 RELATED WORK

Covert channels in the cloud have been known for a long time. Ristenpart et al. (Ristenpart et al., 2009) first exposed cloud computing to covert channel attacks. Authors introduced a technique based on the CPU L2 cache. The L2 cache is utilized to store recently accessed information between the L1 cache and RAM memory. They had implemented a L2 cache covert channel in Amazon EC2. The bit rate was just 0.2 bps. Despite its low bit rate of 0.2bps, this covert channel shows deficiencies in the isolation of virtual machines in Amazon EC2. The limitation of this covert channel is that the sender and receiver need to share the same core. Xu et al. (Xu et al., 2011) refined the communication model of L2 Cache Covert Channel and the channel arrived at considerably higher bandwidth (3.20bps) in EC2. In (Okamura et al., 2010), Okamura et al. designed and evaluated a similar approach which utilized the load of a shared CPU to encode secret data bits. The resulting bandwidth was about 2 bps. This covert channel has little practical applicability as it only works under the assumption that both colluding cloud instances share the same processor's physical core. Except for covert channels using CPU cache, there were some other covert channels using other resources, such as core alternation (Li et al., 2012), sharing memory etc. (Wu et al., 2011) (Shen et al., 2013) (Wu et.al., 2014)

Compared to other cloud covert channels, CCRCVM is used in user-level. As depicted in figure 1, the information flow 1 indicates the VM-level cloud covert channel's flow direction. The flow 2 indicates the user-level cloud covert channel's flow direction. The VM-level covert channel needs to use the VMs to operate the shared resources such as CPU, memory, core etc. Although the user-level covert channel also needs to create VMs, it does not need to run the VMs to operate the shared resources. The user-level covert channel uses the cloud platform managements such as VM create, VM delete and VM migrate to construct the covert channel. The user-level sender and receiver are cloud users rather than virtual machines. Traditional cloud covert channels use VMs as the sender and receiver.

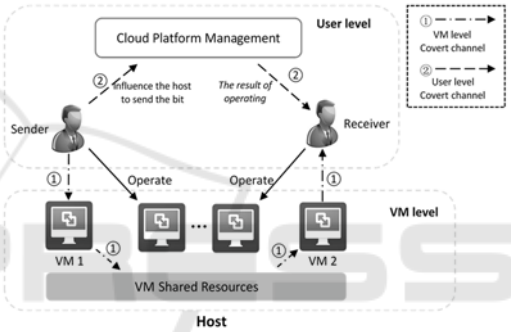

Figure 1: The difference between VM-level and user-level covert channel.

The shared resource that CCRCVM uses is the result of creating a virtual machine which was not mentioned before. The CCRCVM is a cloud user-level covert channel, which means it does not need to operate in virtual machines. This means the monitors cannot find the communications using CCRCVM. The traditional covert channel detections are not effective for CCRCVM. Maybe the bandwidth of CCRCVM is not satisfying, but there is still useful in some conditions. When all traditional ways are monitored in a cloud (Pitropakis et al., 2015), and the malicious user needs to transmit a confident but small amount of information such as the password or key, we should use CCRCVM in this condition.

3 THE SPECIFICATIONS OF CCRCVM

This covert channel can be used by two users in the same IaaS (Infrastructure as a Service) cloud to

communicate. An IaaS user requests to create a virtual machine, and the result may be succeeded or failed. Different creating results implicate different information that gets from other users. For example, in the OpenStack environment, users give a request to create a virtual machine on a specified host. If the host has enough resources, this request is satisfied, and the virtual machine will be the *Active* status; if the host doesn't have enough resources, the virtual machine will be the *Error* status. We can construct our covert channel by this feature.

In this Section, we describe this covert channel in detail, and describe the threat scenario. We also describe the action that sender and receiver need to do. The implementation of CCRCVM will be described in Section 4.

3.1 Threat Scenario

In this section, we introduce the threat scenario of CCRCVM.

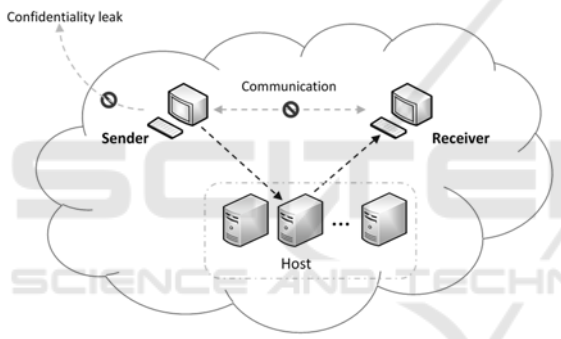

Figure 2: Threat scenario.

As depicted in Figure 2, there are two roles in this threat scenario. One is the sender. The other is the receiver. They are in a common attack organization. They can create VMs in the same IaaS cloud. They cannot communicate in the real world, so they must communicate by the virtual methods. Direct communication is prohibited, and the sender cannot transmit the information outside the cloud. Sender's network packets are monitored. We assume that the sender gets some important information and the sender needs a way to transmit the information outside the cloud to the attack organization. We assume that the cloud provider permits the user to choose the host where the VMs run. The CCRCVM works in this scenario. In reality, this scenario exists in some cloud platforms, such as OpenStack.

3.2 Sender

This covert channel is based on the result of creating a virtual machine. Sender is responsible for influencing the host to send the bit. Figure 3 describes the theory of CCRCVM. We assume the sender can create VMs in a specified host. When the sender sends different bits, the sender needs to behave differently. The actions that sender needs to do are as follows.

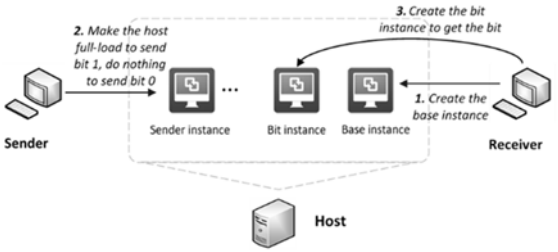

Figure 3: The theory of CCRCVM.

- As depicted in the step 2 of Figure 3, when the sender wants to send bit 1, the sender needs to make the host full-load. Then the host cannot provide the service to create VMs. Sender needs time to wait for the receiver to sense the full-load. After waiting receiver, sender relieve the full-load condition. The time that transmitting a bit consumes is a cycle.
- When the sender sends bit 0, the cycle must equal to the cycle of bit 1. In bit 0 cycle, sender needs to do nothing. The host can provide service to create VMs when sender do nothing. The equal cycle between bit 0 and bit 1 guarantees that every cycle can be received correctly.

3.3 Receiver

This covert channel is based on the result of creating a virtual machine. Receiver is responsible for creating a virtual machine to get the bit. Compared to sender, receiver does not behave differently when it receives different bits. The receiver just needs to create VMs periodically. The actions that the receiver needs to do are as follows.

- As depicted in the step 1 of Figure 3, before the transmit cycle, receiver needs to create a base instance which is a virtual machine to choose a host to construct the covert channel. Receiver should specify an attribute instance group when create base instance. The instance group means that when the VMs are in the same instance group, they are in a same host.

- As depicted in the step 3 of Figure 3, the subsequent VMs are created in a same instance group. Receiver can judge the bit by the result of creating a virtual machine. The result success means transmission of bit 0, and failure means bit 1.

4 THE IMPLEMENTATION OF CCRCVM

In this section, we describe the implementation of CCRCVM and the communication protocol. There are two key points in this implementation, and we will describe these in section 4.1 and 4.2. The hardware and software environments will be described in Section 5.

4.1 Make the Host Full-load

The key to transmit bit using CCRCVM is how to make a host full-load. When a host is full-load, the scheduler cannot choose this host to create a virtual machine. When a user wants to create a virtual machine, the result must be the failure.

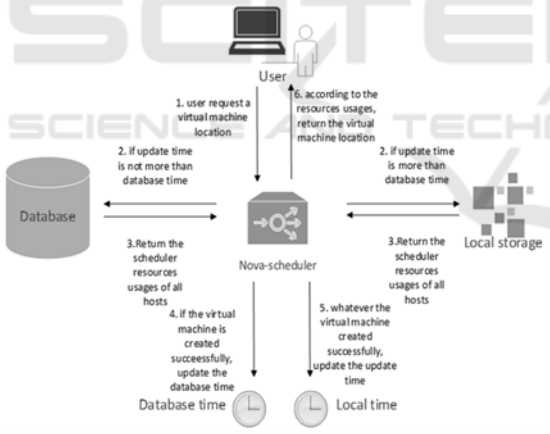

Figure 4: OpenStack schedule process.

How to make the host full-load is important. It has a great influence on the result of CCRCVM. You can create VMs on a specified host until it is full-load. This is a feasible way, but this way is too slow to carry out. In OpenStack environments, we found another way to make a host full-load. This is an easy way to make the host full-load but not the only way.

Nova-scheduler service is responsible for choosing a host for a virtual machine in OpenStack. As depicted in Figure 4, it describes the process of a virtual machine create request in nova-scheduler. In the first step, a user proposes a request which requests to create a virtual machine or some VMs. In the second step, according to the relationship of the resource update timestamp in the database and the resource update timestamp in local, scheduler can decide whether update the resource list or not. At the remainder of this paper, we call the resource update timestamp in database the database time and timestamp in local the local time. If local time is not more than database time, it represents the resource list that stores in local is not the newest. Scheduler should get the resource list from the database. On the contrary, if local time is more than database time, it reveals the resource list in local is latest. The resource list in local is reliable, and scheduler can schedule by referring it. In the step 3, scheduler choose the right resource list to schedule a host. In step 4 and step 5, database and local update their time respectively. There is a difference between them. Only when the request is satisfied, the database time is updated. However, the local time is updated even if the request is not satisfied. In step 6, scheduler returns the location of every VMs or returns the failure information. Users can create more than one VMs in a create request. In the same request, scheduler schedule the VMs sequentially. So when scheduler finishes scheduling a virtual machine, it needs to consume the resource in local. Until all the VMs are scheduled successfully, the resource in the host will be allocated and the database will be updated.

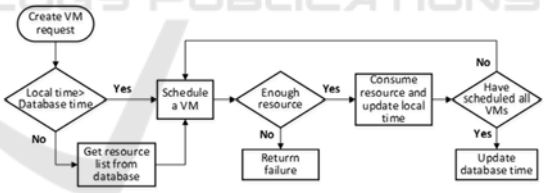

Figure 5: Make the host full-load in OpenStack.

In this paper, the way to make the host full-load is based on the defect of OpenStack. We can find this from the OpenStack source code. The Figure 5 describe this process. When the remaining resources are not enough to satisfy an overload request, the resource list in database is not updated. But the local resource list is updated, and the local time is updated too. This will lead to a vicious circle. When the user sends a create request next time, the local time is more than database time, so scheduler believe the local resource list is latest, and it does not update the local resource list. The database time will not be updated unless the resource in host will be changed (VMs delete action). Because of this, the database time is still less than local time, and the local resource is still

full-load. The following create request will not be satisfied. In short, we can send a request contains overload VMs to make the host still full-load.

4.2 Make the VMs on the Same Host

The other key point is how to create the virtual machine on a specified host. We use an attribute called *server-group* when we create VMs in OpenStack (like this command, nova boot –server-group=group_id). The VMs with the same *server-group* will be created on the same host. Through server-group, receiver should create a base instance to bind the *server-group* to a host. Then, receiver creates VMs with the same *server-group* will be created in the same host which has created the base instance. In this implementation, sender is an OpenStack user who can use the availability-zone attribute when he creates a virtual machine. The user can assign a virtual machine on a specified host using availability-zone attribute. Who can use the availability-zone attribute is depended on the configuration by the platform. Sender should get the location of sender's base instance, and the sender can create an instance with the same host with receiver base instance. Finally, sender should create overload VMs on the same host by *server-group*.

4.3 Communication Protocol

In this section, we describe the protocol of CCRCVM. Sender and receiver need to synchronize before transmit the bit. The communication protocol consists of two phases: the synchronization phase, and the bit transmission phase.

As depicted in Figure 6, it describes the communication protocol of CCRCVM. Receiver gets the bit by the result of creating a VM. However, receiver will get the bit 0 when the sender does not transmit the bit in reality. Therefore we need a symbol to represent the start of the transmission. In our communication protocol, the synchronization phase consists of 3 cycles. If the receiver gets the bit 101 in 3 cycles, we believe the sender is ready to transmit and the bit transmission phase starts.

In bit transmission phase, we define the time that transmitting a bit consumes a cycle. In a cycle, sender and receiver need to do their specified work to transmit the bit. In the first half cycle, Sender needs to change the host state to transmit bit. At the same time, receiver keeps the sleep state. In the second half cycle, sender changes into the sleep state, and receiver captures the bit by creating a virtual machine. Before the transmission phase, receiver needs to create a base instance to bind the *server-group* to a host.

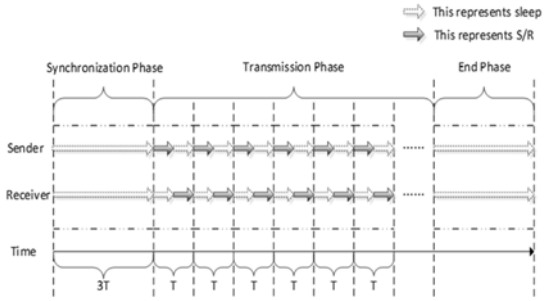

Figure 6: Communication Protocol of CCRCVM.

In a bit cycle, receiver creates a bit instance which has the same *server-group* with the base instance. Then receiver sleeps 5 seconds to wait for the request to finish. Subsequently, receiver gets the result of bit instance. According to the result, receiver can judge the bit is 1 or 0. Finally, receiver deletes the bit instance and sleeps 50s to wait for the next cycle. Receiver repeat these in every bit cycle.

Sender behaves differently when transmitting different bits. If sender transmits bit 1, sender needs to do the following actions. First, sender needs to create a sender base instance on the same host with the receiver base instance and bind a sender *server-group*. Then sender sleeps 5 seconds to wait for the request to finish. Next sender create overload VMs with the same *server-group* with sender base instance. After that, sender enters sleep to give the receiver enough time to get the bit. Finally, sender deletes the base instance and overload instances to relieve the full-load and enter the next cycle. If the sender wants to send bit 0, sender just sleeps a cycle time. In our experiment, the cycle is 60 seconds, and the bandwidth is 0.0167bps.

5 EVALUATION

In this paper, we focus on the existence of the CCRCVM, for this reason, we construct the CCRCVM in a simple environment. This does not influence the existence of the CCRCVM. As long as the sender and receiver satisfy the condition mentioned before, the CCRCVM will exist in a complex environment.

We implemented a CCRCVM prototype on a Lenovo v2000 notebook which has Intel i7-4510U running at 2.00GHz and 8GB RAM. Whole system runs on Windows 8. We run VMware Workstation on this windows host, and create 4 VMs with 2 cores and

2GB RAM. The VMs' operating system is Ubuntu 14.04. We build an OpenStack environment whose version is Juno on the 4 VMware VMs. Our OpenStack environment consists of 1 controller node, 1 network node, and 2 compute nodes. We installed the Keystone, Horizon, Glance and Nova component except nova-compute in Controller node. The controller node is responsible for controlling the system. The Neutron component was installed on the network node. As the name suggests, the network node manages the network of OpenStack. The environment includes 2 compute nodes that were installed nova-compute service. The compute node was the key to create VMs. Only the compute node can provide VMs to users. In aforementioned environment, we used OpenStack users as sender and normal user as receiver.

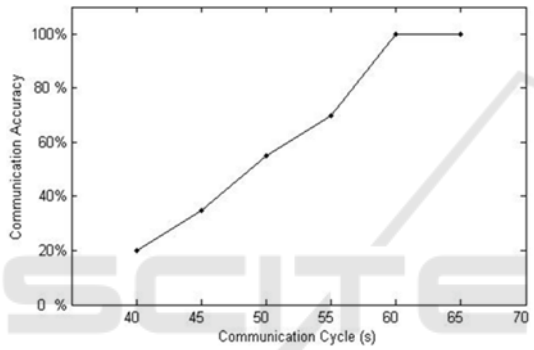

Figure 7: Accuracy of Communication.

In this section, we discuss the relationship between the bandwidth and the accuracy of CCRCVM. These experiments are based on the implementation on section 4. We transmit 50 bits information in one experiment (because the experiment consumes too much time. We choose 50 bits). We choose the bit cycle and we observe the accuracy of transmitting. For every bit cycle, we do the experiments 3 times in every cycle. We conclude the relationship between bandwidth and accuracy.

The results of our experiment are shown in Figure 7, when the bit cycle is 60 seconds, the accuracy of the covert channel is 100%. With the increasing of the bit cycle, the accuracy is increasing. The minimum cycle of the covert channel is 40s, and the accuracy is 20%.

The bandwidth of CCRCVM is not so high in our experiments, but it is possible to be improved in the future. We can use the Markov model for bandwidth computation proposed by Tsai (Tsai et al., 1988) to evaluate the ideal bandwidth of CCRCVM.

$$B=b*(T_r+T_s+2T_{cs})^{-1} \quad (1)$$

The formula 1 is the way to evaluate the ideal bandwidth. The b is code factor, and it is usually 1. The T_r is the time that reading the bit 1 or 0 consumes, and the T_s is the time that setting the bit 1 or 0 consumes. The T_{cs} is the time that configuring the transmit environment consumes. In CCRCVM, T_r, T_s and T_{cs} are depended on the execute time of creating virtual machines. If the environment is good enough, T_r T_s and T_{cs} are possible to be the millisecond level. Therefore, the bandwidth of CCRCVM is possible to be more than 100bps.

6 DISCUSSION

In this section, we discuss the factors influencing the bandwidth of CCRCVM, the influence of other users and the elimination of CCRCVM.

In our environment, when the accuracy is 100%, the bandwidth is only 0.0167bps. The reason for that is as follows. First, regardless of the sender or the receiver, when we input the creating command, the command needs time to execute. This time is usually 1-2s in our environment. Second, when the command is finished, we need to give some time to compute node to launch the virtual machine. That is the reason why we sleep after inputting the creating command. Third, when the sender wants to transmit bit 1, it should make the host full-load. The full-load status should keep a period time, which makes an extra time overhead. The first two reasons are about the execution effect. So when we use CCRCVM in a better environment, the bandwidth must be larger. The third reason is an effect problem actually. If the sender can make the host full-load faster and the receiver gets the bit faster, the time to remain must be slower. The bandwidth must be larger.

Because the scheduler service for the request in order, other requests cannot influence the CCRCVM. When CCRCVM is running，any user cannot get the timely schedule service. Thus, the faster the CCRCVM run, the more difficult the CCRCVM will be found.

There are two key points in CCRCVM. One is how to make the host full-load, and the other is how to make the virtual machines on the same host. We can propose the elimination from these two points. For full load, we can restrict the number of virtual machines that can be created on the same host to eliminate the CCRCVM. For the location of VMs, we can forbid the choice to create a virtual machine on a specified host regardless admin user or the normal user. However, these two ways are not easy to eliminate the covert channel.

The tradeoff between the VMs threshold given to the user and the elimination of this covert channel is significant. If the platform makes a threshold too small, the users cannot create enough virtual machines to complete their tasks. However, if the threshold is not small enough, the CCRCVM cannot be eliminated. For example, the sender can choose a kind of virtual machine which occupies a large amount of resources. The sender just create a few VMs (less than the threshold) and can make the host full-load. The sender VMs threshold is not worked in this condition.

If the cloud platform prohibits the capability that a user can choose the location of a virtual machine absolutely. The management of the cloud platform will be not flexible. For example, a safety sensitive client needs a complete isolation environment, because of no privileged user, the request cannot be satisfied. However, once a user is authorized to choose the host. The threat of CCRCVM will appear, because every user is not be guaranteed safety. In reality, OpenStack permits the user to choose the host.

7 CONCLUSIONS

In this paper, first, we propose a new covert channel CCRCVM which can make two users to communicate in IaaS cloud. Next we describe the threat of CCRCVM and the theory of the sender and receiver. We implement a prototype of CCRCVM in OpenStack environment. We communicate successfully between two OpenStack users by CCRCVM. In our prototype. We do many experiments. The results show that the accuracy is 100% when the bit cycle is 60s. In this situation, the bandwidth is 0.0167bps.

In this paper, we present CCRCVM, an user-level covert channel. We believe some other user-level covert channels should exist, such as using the result of VMs migration to construct the covert channel. In future, we will continue to do the research of these user-level covert channels.

ACKNOWLEDGEMENTS

This work is supported by the National High Technology Research and Development Program (“863” Program) of China under Grant No. 2015AA016009, the National Natural Science Foundation of China under Grant No. 61232005, and the Science and Technology Program of Shen Zhen, China under Grant No. JSGG20140516162852628.

REFERENCES

Alarifi, S. S., & Wolthusen, S. D. (2012, December). Detecting anomalies in IaaS environments through virtual machine host system call analysis. In *Internet Technology And Secured Transactions*, 2012 International Conference for (pp. 211-218). IEEE.

Bijon, K., Krishnan, R., & Sandhu, R. (2015, June). Mitigating Multi-Tenancy Risks in IaaS Cloud Through Constraints-Driven Virtual Resource Scheduling. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies* (pp. 63-74). ACM.

Han, Y., Chan, J., Alpcan, T., & Leckie, C. (2014, June). Virtual machine allocation policies against co-resident attacks in cloud computing. In *Communications (ICC), 2014 IEEE International Conference on* (pp. 786-792). IEEE.

Jaeger, T., Sailer, R., & Sreenivasan, Y. (2007, June). Managing the risk of covert information flows in virtual machine systems. In *Proceedings of the 12th ACM symposium on Access control models and technologies* (pp. 81-90). ACM.

Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, 16(10), 613-615.

Li, Y., Shen, Q., Zhang, C., Sun, P., Chen, Y., & Qing, S. (2012, March). A covert channel using core alternation. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on* (pp. 324-328). IEEE.

Meade, F. G. G. (1993). A guide to understanding covert channel analysis of trusted systems. NCSC4TG4030 National computer security center, Maryland university.

Okamura, K., & Oyama, Y. (2010, March). Load-based covert channels between Xen virtual machines. In *Proceedings of the 2010 ACM Symposium on Applied Computing* (pp. 173-180). ACM.

Percival, C. (2005). Cache missing for fun and profit.

Pitropakis, N., Lambrinoudakis, C., & Geneiatakis, D. (2015). Till All Are One: Towards a Unified Cloud IDS. In *Trust, Privacy and Security in Digital Business* (pp. 136-149). Springer International Publishing.

Reuben, J. S. (2007). A survey on virtual machine security. Helsinki University of Technology, 2, 36.

Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009, November). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security* (pp. 199-212). ACM.

Shah, G., Molina, A., & Blaze, M. (2006, July). Keyboards and Covert Channels. In *USENIX Security*.

Shen, Q., Wan, M., Zhang, Z., Zhang, Z., Qing, S., & Wu, Z. (2013). A covert channel using event channel state

on xen hypervisor. In *Information and Communications Security* (pp. 125-134). Springer International Publishing.

Tsai, C. R., & Gligor, V. D. (1988, April). A bandwidth computation model for covert storage channels and its applications. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on* (pp. 108-121). IEEE.

Varadarajan, V., Zhang, Y., Ristenpart, T., & Swift, M. (2015, August). A placement vulnerability study in multi-tenant public clouds. In *24th USENIX Security Symposium (USENIX Security 15)(*Washington, DC (pp. 913-928).

Vleck, T. V. (1990). Timing channels. Poster session. In *IEEE TCSP conference.*

Wu, J., Ding, L., Wang, Y., & Han, W. (2011, July). Identification and evaluation of sharing memory covert timing channel in Xen virtual machines. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on* (pp. 283-291). IEEE.

Wu, Z., Xu, Z., & Wang, H. (2014). Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud.

Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., & Schlichting, R. (2011, October). An exploration of L2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (pp. 29-40). ACM.